

# The Structure Of SAT

Murthy VSN Oruganty

April 17, 2014

## Abstract

The general structure of the unrestricted Boolean Satisfiability problem in Conjunctive Normal Form (CNF) is presented using elementary logic. A clause-tree generated by the reduction of the largest length root clauses (RCs) to smaller ones is utilized to compute the satisfiability of the exhaustive  $2^{3^n-1}$  Propositional Formulas (PFs) from the  $3^n - 1$  legitimate non-null clauses for  $n$  variables. The number of PFs increases monotonically and approximately in a geometric progression in  $2^n$  as their ‘degree of satisfiability’ changes from tautology to unsatisfiable. This fact is demonstrated for low  $n$  along with derived closed-form expressions for such numbers of PFs that remain valid for higher  $n$ . It is also shown that all unsatisfiable PFs can be transformed using elementary operations to a Basic Contradiction Clause (BCC). The time and space complexities of such a transformation are directly related to the number of clauses and literals in their core(s).

## 1 Introduction

In practical computing, complexity<sup>1</sup> is at least as significant as decidability<sup>2</sup>. Satisfiability is the problem of deciding if there exists an assignment to the Boolean variables in the given propositional formula (PF) that evaluates it to TRUE. Satisfiable instances can be proven to be satisfiable by algorithmically arriving at a satisfying assignment, failing which the PF is termed

---

<sup>1</sup>S. Cook (1971). “The complexity of theorem proving procedures”. Proc. of the Third Annual ACM Symposium on Theory of Computing: pp. 151-8. and L. Levin (1973) (in Russian). “Universal search problems” Problems of Information Transmission **9** (3): pp. 115-6. Cited in M. Sipser. (2005) “Introduction To The Theory of Computation” Sec 7.4

<sup>2</sup>A. M. Turing (1937) “On computable numbers, with an application to the Entscheidungsproblem”. Proc. of the London Mathematical Society **2** 42: pp. 230-65

intractable: the algorithm<sup>3</sup> is unable to decide within the allocated resources of time and space. The only way other than exhaustive search to prove a PF unsatisfiable is by way of logical contradiction.

Using unsatisfying assignments as the basis, we shall construct the clause-tree, define a few quantities and operations, and apply the construction to arrive at some results on 1) inverse #SAT problem and 2) proving unsatisfiability of PFs.

## 2 Basics

A Boolean variable  $(x_1, x_2, x_3 \dots x_n)$  is a variable taking on a value of either TRUE (1) or FALSE (0). A Boolean variable or its complement is a literal  $(x_1, \text{and } \neg x_1 \text{ or } x'_1)$ . A group of  $k$  literals connected via disjunction (OR operator  $\vee$ ) is a length  $k$ -clause, e.g.  $c = (x_1 \vee x_2 \vee \neg x_3)$  or  $c = x_1 x_2 x'_3$  such that  $\|c\| = 3$ . A conjunction (AND operator  $\wedge$ ) of clauses forms a propositional formula (PF) in CNF, e.g.  $f = \bigwedge_{i=1}^m c_i$  or the corresponding set

$F = \{c_1, c_2, c_3, \dots, c_m\}$  having  $m$  clauses and  $l = \sum_{i=1}^m \|c_i\|$  literals. A PF that has at least one satisfying assignment out of the possible  $2^n$  as a solution, i.e. evaluating to TRUE, is satisfiable and unsatisfiable otherwise. Two PFs that satisfy the same assignments in  $n$  variables are equivalent logically ( $\Leftrightarrow$ ).  $\mathbb{C}$  is the universal set of all possible clauses for a given  $n$ ,  $|\mathbb{C}| = 3^n$ .

## 3 The Clause-tree: operations, transformations and logical equivalence

PF  $f$  (or set  $F$ ) can be transformed to  $g$  ( $G$ ) using the following constructs among others.

**Definition 1.** The universal set of *Root Clauses*  $\mathbb{R} = \{r : r \in \mathbb{C} \text{ and } \|r\| = n\}$  for  $n$  variables.

*Remark.* Root Clauses (RCs) are unit constraints having a one-one correspondence with FALSE assignments, (analogous to an orthonormal basis).  $2^n$  RCs exist for given  $n$ . Using this principle, every PF  $f = \bigwedge_{i=1}^m c_i$  or its set  $F = \{c_1, c_2, c_3 \dots, c_m\}$  has a unique logical equivalent in  $\mathbb{R}$  henceforth called

---

<sup>3</sup>such as variants of M. Davis and H. Putnam (1958). “Computational methods in the propositional calculus”. *Cited in* John Franco and John Martin. “A history of satisfiability” in Handbook of Satisfiability, Armin Biere, Marijn Heule, Hans van Maaren, Toby Walsh (eds.), p 19. IOS Press, 2009

its root equivalent,  $\rho_f = \bigwedge_{i=1}^{|R_f|} r_i$  where  $r_i \in R_f = R_F = \bigcup_{i=1}^m R_{c_i}$  such that the  $r_i$  do not satisfy the same assignments that  $f$  does not.  $R_{c_i} \subset \mathbb{R}$  and  $R_f = R_F \subseteq \mathbb{R}$ .

**Lemma 3.1.** For any two PFs  $f$  and  $g$ ,  $R_F = R_G \iff f \Leftrightarrow g$ .

*Remark.* The root equivalent (a PF representation of the truth table) thus defines the logical content of a PF. As a corollary,  $R_f = \{\emptyset\}$  for the tautological null PF  $f = \emptyset$ , and  $R_{\rho_{\perp}} = \mathbb{R}$  for the unsatisfiable Full RCPF

$$\rho_{\perp} = \bigwedge_{i=1}^{2^n} r_i.$$

**Definition 2.** An *expansion* ( $\triangleleft$ ) operation on a clause  $c \in F$  using  $x$  results in a logically equivalent clause pair of *predecessors*,  $\{c\} \triangleleft \{(c \vee x), (c \vee \neg x)\}$  or compactly  $\triangleleft(c, x \rightarrow c \vee x, c \vee \neg x)$ . The inverse of an expansion operation, a *reduction* ( $\triangleright$ ) operation on a clause-pair in  $F$ ,  $\{(c \vee x), (c \vee \neg x)\} \triangleright \{c\}$  eliminates  $x$  to yield their logically equivalent common *successor* denoted compactly  $\triangleright(c \vee x, c \vee \neg x, x \rightarrow c)$ .  $R_{\{(c \vee x), (c \vee \neg x)\}} = R_{\{c\}}$ .

**Definition 3.** An *insertion* ( $\nabla(c, x \rightarrow c \vee x)$ ) operation on a clause  $c \in F$  using  $x$  results in its predecessor  $(c \vee x)$ . The inverse, a *literal drop* ( $\Delta(c \vee x, x \rightarrow c)$ ) operation of  $x$  on a clause  $(c \vee x) \in F$  results in its successor  $c$ .  $R_{c'} \subsetneq R_c$  and  $|R_{c'}| = \frac{|R_c|}{2}$ .

**Definition 4.** An *introduction of a repetition*  $\Downarrow (c \rightarrow c1 = c, c2 = c)$  or *removal of a repetition*  $\Uparrow (c1 = c, c2 = c \rightarrow c)$  are logically equivalent operations that create or remove a duplicate copy of a clause in a PF or its set equivalent.

*Remark.* Complexity<sup>4</sup>. It is assumed that each of the above logical operations takes a unit step in time to execute, i.e.  $T_{\triangleleft} = T_{\triangleright} = T_{\nabla} = T_{\Delta} = T_{\Downarrow} = T_{\Uparrow} = +1$ , and space required for the operations in terms of cardinality (all clauses occupying same space) is  $S_{\triangleleft} = +1$ ,  $S_{\triangleright} = -1$ ,  $S_{\nabla} = 0$ ,  $S_{\Delta} = 0$ ,  $S_{\Downarrow} = +1$ ,  $S_{\Uparrow} = -1$ . The number of literals introduced by the operations are  $L_{\triangleleft} = \|c\| + 2$ ,  $L_{\triangleright} = -(\|c\| + 2)$ ,  $L_{\nabla} = +1$ ,  $L_{\Delta} = -1$ ,  $L_{\Downarrow} = \|c\|$ ,  $L_{\Uparrow} = -\|c\|$ . For a transformation  $f \xrightarrow{\mathfrak{S}} g$  through a path-dependent sequence  $\mathfrak{S}$  of

$$\text{length (number of logical operations) } |\mathfrak{S}| = T, \quad |G| - |F| = \sum_{i=1}^T S_{\mathfrak{S}_i}, \quad L_g - L_f = \sum_{i=1}^{m_g} \|c_i \in G\| - \sum_{i=1}^{m_f} \|c_i \in F\| = \sum_{i=1}^T L_{\mathfrak{S}_i} \text{ calculated sequentially, but } T_{tot}(f, g, \mathfrak{S}) =$$

<sup>4</sup>equivalent in polynomial sense to the time and space notions in J. Hartmanis and R. E. Stearns (1965). "On the computational complexity of algorithms". Transactions of the American Mathematical Society **117**: pp 285-306.

$\sum_{i=1}^T T_{\mathfrak{S}_i} = T$ ,  $S_{max}(f, g, \mathfrak{S})$  determine the time and space required to successfully perform it.

It is worth noting that  $\mathfrak{S}[\mathfrak{S}_1 = \triangleright(c \vee x, c \vee \neg x, x \rightarrow c)] = \mathfrak{S}[\mathfrak{S}_1 = \Delta(c \vee x, x \rightarrow c), \mathfrak{S}_2 = \Delta(c \vee \neg x, x \rightarrow c), \mathfrak{S}_3 = \Uparrow(c(\mathfrak{S}_2))]$  and similarly  $\mathfrak{S}[\mathfrak{S}_1 = \triangleleft(c, x \rightarrow c \vee x, c \vee \neg x)] = \mathfrak{S}[\mathfrak{S}_1 = \Downarrow(c \rightarrow c1 = c, c2 = c), \mathfrak{S}_2 = \nabla(c1, x \rightarrow c \vee x), \mathfrak{S}_3 = \nabla(c2, x \rightarrow c \vee \neg x)]$  amongst other possibilities.

**Definition 5.** The Basic Contradiction Clause (BCC, denoted by  $c_\perp$  or  $()$ ) defined by  $\|c_\perp\| = 0$ , having the following properties:

- In contrast to the null clause or PF  $\emptyset$ , denoting absence of any constraint and that satisfies all assignments, BCC represents all possible constraints and renders the whole CNF PF unsatisfiable.
- Can be formed from any clause by dropping all its literals or as a result of reduction of  $x_l \wedge \neg x_l$ .

$$\bullet |R_{c_\perp}| = 2^n, \quad c_\perp \Leftrightarrow \bigwedge_{i=1}^{2^n} r_i = \rho_\perp, \quad R_{c_\perp} = R_{\rho_\perp} = \mathbb{R}.$$

**Definition 6.** A *clause-tree* for  $n$  variables (such as Figure 1) can be constructed by either expanding fully from BCC using any of the available variables at any stage or by successively reducing clauses of  $\rho_\perp$  by eliminating each variable until BCC is reached.

**Lemma 3.2.** *BCC can be logically equivalently expanded exhaustively to  $2^l$  root clauses of any subspace of  $l \leq n$  variables (branches of the expansion realization of the clause-tree of Definition 6) or reduced from them (reduction realization) in a minimum number of operations  $T^{min} = 2^l - 1$ .*

*Proof.* Choose a variable, say  $x_1$ . Introduce (eliminate)  $x_1$  by expanding (reducing pairwise neighbours) the null clause (given  $2^l$  clauses in dictionary order). The number of operations required is  $2^{l-1}$ . Similarly, expanding(reducing) successively using single variables at each stage the minimum number of operations required for the transformations are  $T_\triangleleft(T_\triangleright) = 2^{l-1} + 2^{l-2} \dots + 2^0 = 2^l - 1$  respectively.  $\square$

## 4 Inverse #SAT

**Definition 7.** The degree of satisfiability of  $f$ ,  $q = 2^n - |R_f| = 2^n - r$ .

The number of PFs amongst all the  $2^{3^n-1}$  PFs for  $n$  variables having the same degree of satisfiability  $q$ ,  ${}^n\#_r$ , can be evaluated by counting the number of logically equivalent ways of choosing  $r$  RCs and all of their possible successors *obtained by reduction among themselves* and is tabulated for  $n =$

2, 3 in Table 1. Diagonal closed-form expressions for  ${}^n\#_{r=n}$  remain valid for higher  $n$ .

## 5 Unsatisfiability

**Lemma 5.1.** *Any unsatisfiable PF  $f \Leftrightarrow \rho_\perp$  and  $R_f = R_F = \mathbb{R}$ .*

*Proof.* The proof follows from Lemma 3.1.  $\rho_f = \bigwedge_{i=1}^{|R_f|} r_i$  where  $r_i \in R_f = \bigcup_{i=1}^m R_{c_i} = \mathbb{R}$ .  $\square$

**Lemma 5.2.** *For a PF  $f$ , if  $R_F = \mathbb{R}, \exists \tilde{F} \subset F$  such that  $R_{\tilde{F}} = \mathbb{R}$  and  $\forall c_i \in \tilde{F}, R_{\tilde{F}-\{c_i\}} \neq \mathbb{R}$ .*

*Remark.* For the proof,  $F \subseteq \tilde{F}$  ensures existence of  $\tilde{f}$ .  $\tilde{f}$  is a minimal, self-sufficient unsatisfiable subset *core* of clauses of  $f$ . All other clauses in  $f$  are *extraneous* to  $\tilde{f}$ . A theoretical procedure to ascertain a core of  $f$ : let  $\tilde{F} = F - c_1$ . If  $\tilde{f} \Leftrightarrow \rho_\perp$ , assign  $\tilde{F}$  to  $F$ . Else perform the check on the next clause  $c_2$ . After checking for each such clause  $c_i \in F$ , the residual PF  $\tilde{f}$  is the required expression.

**Theorem 5.3.** *For any core  $\tilde{f}$  of length  $m$  clauses and  $l$  literals,  $\exists(\{c_\perp\} \xrightarrow{\mathfrak{S}} \tilde{F})$  with (i)  $\mathfrak{S}$  consisting only of  $\triangleleft, \Delta, \uparrow$  operations and (ii)  $|\mathfrak{S}| = f(m, l)$  a non-exponential function.*

*Proof.* (i) Maximally expand  $c_\perp$  to Full RCPF  $\rho_\perp$ . Since  $\|r_i\| = n \forall r_i \in \mathbb{R}$ , find a predecessor amongst  $r_i$ s  $\forall c_i \in \tilde{f}$ , drop all the literals other than those in  $c_i$ . When all the core clauses are formed, drop literals suitably from remaining RCs to generate repeats of core clauses. Remove the repeats.

(ii) Assume the transformation  $(\{c_\perp\} \xrightarrow{\mathfrak{S}} \tilde{F})$  requires  $T_\triangleleft$  expansions and  $T_\Delta$  drops. Also assume  $T_\uparrow = a_1 T_\Delta$  repeat removals through reductions or accidental repeats. Also assume the transformation through  $\mathfrak{S}$  has no set of operations that reverse the entire effect of a previous operation, a condition that the minimal transformation also obeys.  $|\mathfrak{S}| = T = T_\triangleleft + (1 + a_1)T_\Delta$ .  $m = 1 + T_\triangleleft - a_1 T_\Delta$ . For each  $c_i$ , its number of literals  $l_i = T_\triangleleft^i - T_\Delta^i$  where  $T_\triangleleft^i, T_\Delta^i$  are the subset of operations  $T_\triangleleft - T_\Delta$  that affect  $c_i$ . Thus  $l = a_2 T_\triangleleft - a_3 (1 - a_1) T_\Delta$  for  $l > 0$  with  $a_2, a_3$  some constants.  $0 < a_2 < m$  and  $0 \leq a_3 < m$  since exceeding the upper bound would mean implausibly that all the expansions and drops affect all the clauses. Solving,  $T = \frac{[(m-1)(a_3(1-a_1)+a_2(1+a_1))-l(1+2a_1)]}{(a_3(1-a_1)-a_2a_1)}$ .  $\square$

*Remark.* Full RCPF  $\rho_\perp$  is the largest possible core. Repeats bifurcate cores

and are thus removed.

In general, expanding from  $c_\perp$  and dropping literals (preserving unsatisfiable assignments) while removing repeat or predecessor clauses generates a core.

**Theorem 5.4.** *Any unsatisfiable PF  $f$  can be transformed to  $c_\perp$  in number of steps a non-exponential function of  $m, l$ .*

*Proof.*  $\tilde{f} = f$  is the largest possible core of  $m$  clauses and  $l$  literals. Thus for any core  $\tilde{f}$  of  $f$ ,  $\exists(\{c_\perp\} \xrightarrow{\mathfrak{S}} \tilde{F})$  with  $T = \frac{[(m-1)(a_3(1-a_1)+a_2(1+a_1))-l(1+2a_1)]}{(a_3(1-a_1)-a_2a_1)}$  using

Theorem 5.3. Retrace the steps of the sequence  $\mathfrak{S}$  with  $(\tilde{F} \xrightarrow{\mathfrak{S}'} \{c_\perp\})$  using an expansion for each dropped literal and a reduction for each expansion using the same variables. The result in  $T$  steps is BCC in conjunction with excess clauses from the expansions in  $\mathfrak{S}'$  of the literal drops in  $\mathfrak{S}$ .  $\square$

*Remark.* The bounds are non-optimal and the question of possible singularities remains to be addressed. The theorem holds, with a slightly different  $T$ , even if only expansions and literal drops are considered ignoring removal or introduction of repetitions as legitimate operations. The non-exponential operations of reading, writing, sorting and searching have also been ignored. In practice, deciding an arbitrary unsatisfiable PF involves 1) identifying the core 2) guessing the dropped literals, both of which may not be tractable.

---

I wish to acknowledge the other works that I could not procure/read/comprehend completely but which nonetheless contributed in the development of this report.

Table 1:  ${}^n\#_r$  for  $n = 2, 3$  and closed-form expressions<sup>a</sup>.

| <b>r</b> | ${}^n\#_r(2^{nr})(\frac{{}^n\#_r}{{}^n\#_{r-1}})$ |                     |                           |     |                                         |
|----------|---------------------------------------------------|---------------------|---------------------------|-----|-----------------------------------------|
|          | n=1                                               | n=2                 | n=3                       | ... | n                                       |
| 0        | 1(1)                                              | 1(1)                | 1(1)                      |     | 1                                       |
| 1        | <b>2</b> (2)                                      | 4(4)(4)             | 8(8)(8)                   |     | $2^n$                                   |
| 2        | 1(4)                                              | <b>22</b> (16)(5.5) | 76(64)(9.5)               |     | ${}^{2^n}C_2 + 4n2^{n-1}$               |
| 3        | -                                                 | 68(64)(3.09)        | <b>536</b> (512)(7.05)    |     | ${}^{2^n}C_3 + 4n2^n(2^{n-1} + n - 2)$  |
| 4        | -                                                 | 161(264)(2.37)      | 4950(4096)(9.23)          |     |                                         |
| 5        | -                                                 | -                   | 36472(32768)(7.37)        |     |                                         |
| 6        | -                                                 | -                   | 362092(262144)(9.93)      |     |                                         |
| 7        | -                                                 | -                   | 3337704(2097152)(9.22)    |     |                                         |
| 8        | -                                                 | -                   | 63367025(16777216)(18.98) |     |                                         |
| $\sum$   | 4(7)                                              | 256(349)            | 67108864(19173961)        |     | $2^{3n-1} (\frac{2^{n2^n+1}-1}{2^n-1})$ |

<sup>a</sup>Corrected  $r = 3$  expression above and

$$\begin{aligned}
{}^n\#_4 = & {}^{2^n}C_4 + 4n2^{n-1}(2^n - 3)(2^{n-1} + 2n - 3) + 8n2^{n-1}(n2^{n-1} - 2n + 1) \\
& + \frac{16}{3}n(n-1)(4n-5)2^{n-1} + 8(1 + 16\theta_{n,3})n(n-1)2^{n-2}, \quad \theta_{i,j} = \begin{cases} 1, & \text{if } i \geq j \\ 0, & \text{otherwise} \end{cases}
\end{aligned}$$

by V. Venkataraman, Indian Institute of Science.

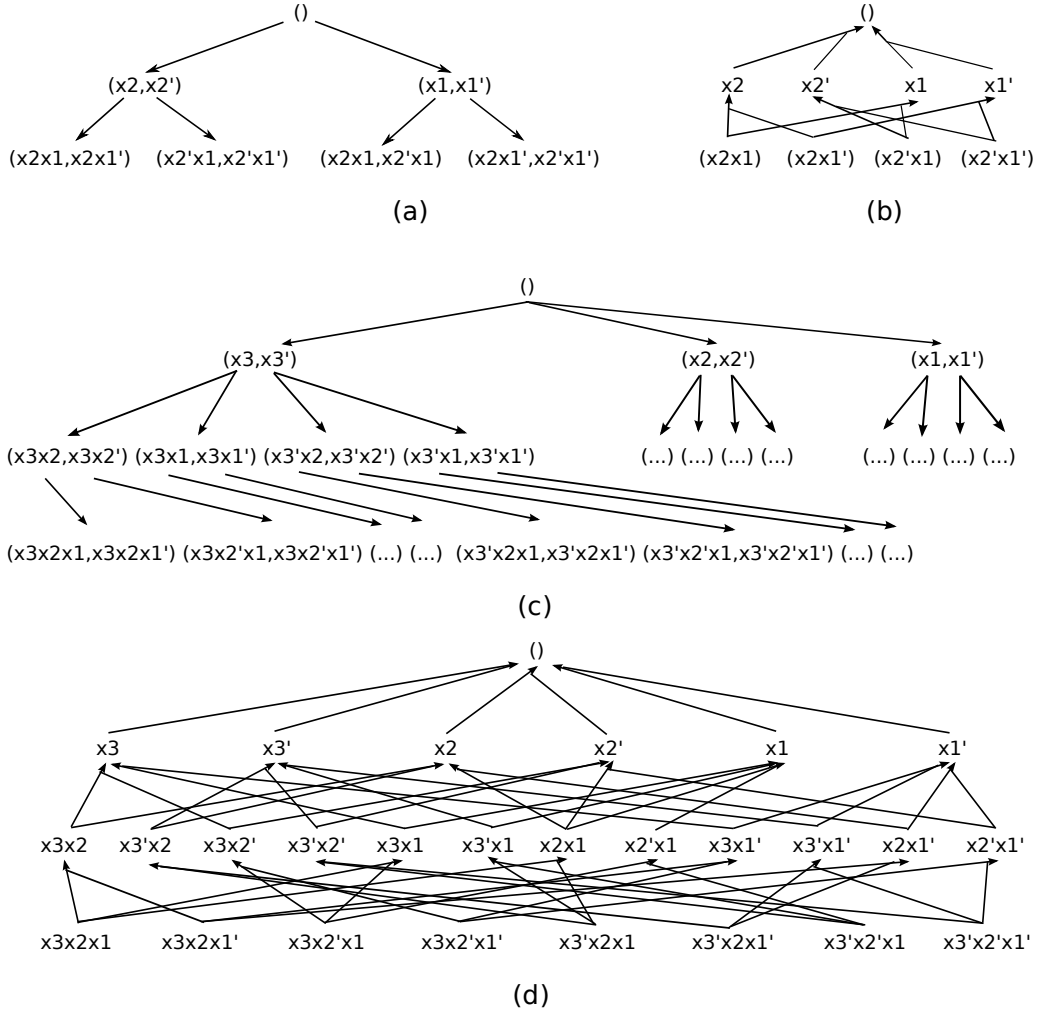


Figure 1: A Clause-tree can be generated from expansion operations from a basic contradiction to root clauses and vice-versa using reduction operations. a) Clause tree for  $n = 2$  by expansion b) Alternative reduction realization of clause tree for  $n = 2$  c) Clause tree for  $n = 3$  by expansion d) Reduction clause tree for  $n = 3$